1 Einleitung

Die *binomialen Heaps* sind eine höhere Datenstruktur, die Operationen auf dynamischen Mengen zuläßt. Sie unterstützen wie die *binären Heaps* zahlreiche Operationen wie das Einfügen oder Löschen von Knoten, das Verringern von Schlüsseln etc. Darüber hinaus sind *binomiale Heaps* in logarithmischer Zeit fusionierbar, was sie gegenüber den *binären Heaps* auszeichnet, wo diese Operation in $\Theta(n)$ läuft.

2 Binomiale Bäume

Ein *binomialer Heap* besteht aus sogenannten *binomialen Bäumen*. Ein *binomialer Baum* ist ein rekursiv definierter geordneter Baum. Der *binomiale Baum* B_0 besteht aus einem einzigen Knoten. Ein *binomialer Baum* B_k besteht aus zwei *binomiale Bäumen* B_{k-1}, wobei die Wurzel des einen das linke Kind der Wurzel des anderen ist.

2.1 Eigenschaften binomialer Bäume

Für einen *binomialen Baum* B_k gilt:

1. Er besteht aus 2^k Knoten
2. Die Höhe des Baumes ist k
3. In der Tiefe $i, (i = 0, 1, ..., k)$ gibt es genau $\binom{k}{i}$ Knoten
4. Die Wurzel hat den größten Grad im Baum mit k. Das i.te Kind ist, wenn die Kinder der Wurzel von B_k von links nach rechts mit $k-1, k-2, ..., 0$ durchnummeriert werden, die Wurzel des Teilbaums B_i.

2.2 Beweis

Der Beweis erfolgt über vollständige Induktion über k. Die Eigenschaften des Lemmas sind für den Induktionsanfang, gebildet durch B_0, trivialerweise erfüllt. Für den Induktionsschritt nehmen wir an, daß die jeweilige Eigenschaft des Lemmas für B_{k-1} gilt.

1. B_k besteht per Definition aus zwei *binomialen Bäumen* B_{k-1}. Laut Induktionsvoraussetzung haben diese jeweils 2^{k-1} Knoten. B_k hat somit $2^{k-1} + 2^{k-1} = 2^k$ Knoten.
2. Wir betrachten das linke Kind der Wurzel von B_k. Dieses ist die Wurzel eines Baums B_{k-1} und hat laut Induktionsvoraussetzung Tiefe $k-1$. Der linke Teilbaum B_{k-1} ist der tieferhängende in B_k, wenn wir noch die Kante zum Wurzelknoten hinzuzählen, haben wir die Gesamttiefe von B_k abgedeckt. Somit hat B_k Tiefe $(k-1)+1 = k$.
3. Bezeichne $D(k, i)$ die Anzahl der Knoten eines *binomialen Baumes* B_k in der Tiefe i. Wenn wir in einem *binomialen Baum* B_k alle Knoten in Tiefe i zählen wollen, dann ergibt sich die Anzahl der Knoten dieser Tiefe aus der Summe der Knoten der beiden Teilbäume B_{k-1}. Der rechte B_{k-1} hat dabei $D(k-1, i)$ Knoten, der linke, der um eine Kante tieferhängt $D(k-1, i-1)$. Nach *Induktionsvoraussetzung* ergeben sich daraus $D(k-1, i) + D(k-1, i-1) = \binom{k-1}{i} + \binom{k-1}{i-1} = \binom{k}{i}$ viele Knoten für B_k in Tiefe i.

4. Die Wurzel von B_k ist gleichzeitig die Wurzel des rechten B_{k-1}. Laut Induktionsvorraussetzung hat diese Grad $k-1$. Hinzu kommt die abgehende Kante zum linkesten Kind, das die Wurzel vom 2. B_{k-1} ist. Somit hat B_k Grad $(k-1)+1=k$. Für die zweite Teileigenschaft betrachten wir B_{k-1}. Laut Induktionsvoraussetzung sind dessen Kinder von rechts nach links betrachtet die Wurzeln der *Binomialbäume* B_0 bis B_{k-2}. Hängen wir an die Wurzel von B_{k-1} nun als linkes Kind erneut einen Baum B_{k-1} so erhalten wir einen *binomialen Baum* B_k, dessen Kinder von rechts nach links die Wurzeln der *Binomialbäume* B_0 bis B_{k-1} sind.

2.3 Korollar

Der maximale Grad jedes Knotens in einem binomialen Baum B_k mit n Knoten ist $\lfloor \lg n \rfloor$. Nach Eigenschaft 1 aus dem Lemma gilt $n = 2^k$, nach Eigenschaft 4, daß die Wurzel von B_k den größten Grad mit k hat. Außerdem gilt $k = \lg 2^k = \lg n$.

3 Binomiale Heaps

Ein binomialer Heap H ist eine Menge binomialer Bäume mit folgenden Eigenschaften:

1. Jeder binomiale Baum erfüllt die *Min-Heap-Eigenschaft*, d.h. der Schlüssel jedes Knotens im Binomial-Baum ist größer oder gleich dem Schlüssel seines Vaterknotens. Daraus folgt, daß der Knoten mit minimalen Schlüssel in der Wurzel liegt.

2. Für jede nichtnegative Zahl k gibt es höchstens einen binomialen Baum B_k in H.

Aus 2.) folgt, daß wenn H n Knoten hat, daß er dann aus höchstens $\lfloor \lg n \rfloor + 1$ binomialen Bäumen besteht. Dies ist daraus ersichtlich, wenn man die Zahl n als Binärzahl $\langle b_{\lfloor \lg n \rfloor}, b_{\lfloor \lg n \rfloor - 1}, ..., b_0 \rangle$ codiert, mit $n = \sum_{i=0}^{\lfloor \lg n \rfloor} b_i 2^i$. Aus Eigenschaft 1 des Lemmas wissen wir, daß ein binomialer Baum B_i genau 2^i Knoten hat. Ein binomialer Baum B_i erscheint genau dann in H, wenn Bit $b_i = 1$ gilt. Deshalb enthält ein binomialer Heap H höchstens $\lfloor \lg n \rfloor + 1$ binomiale Bäume.

3.1 Darstellung binomialer Heaps

Die binomialen Bäume eines binomialen Heaps H sind in einer sogenannten *Wurzelliste* organisiert. Die *Wurzelliste* enthält die Wurzeln der binomialen Bäume von links nach rechts streng monoton steigend sortiert nach Wurzelgrad. Aus den vorigen Überlegungen folgt, daß die Grade der Wurzeln Teilmengen von $0, 1, ..., \lfloor \lg n \rfloor$ sind. Jeder Knoten x eines binomialen Baumes verwaltet neben dem Schlüsselattribut sowie anderen Satellitenattributen Zeiger $p[x]$ auf den Vaterknoten, $linkes-kind[x]$ auf das linkeste Kind sowie $rechter-bruder[x]$ auf den unmittelbar rechten Bruderknoten. Für die Knoten der *Wurzelliste* zeigen die $rechter-bruder$-Zeiger auf die Wurzel des Binomialbaums mit nächsthöherem Grad. Der rechteste Wurzelknoten zeigt auf NIL. Außerdem werden für jeden Knoten x mit dem Attribut $grad[x]$ die Anzahl der Kinder angegeben. Der Heap H verwaltet außerdem noch einen Zeiger $kopf[H]$ auf den ersten Wurzelknoten in der *Wurzelliste*.

3.2 Operationen

3.2.1 Erzeugung eines neuen binomialen Heaps

Um einen leeren binomialen Heap zu erzeugen, wird über die Prozedur *MAKE-BINOMIAL-HEAP* ein Objekt H allokiert und zurückgegeben, $head[H]$ zeigt dabei auf NIL, die Laufzeit ist $\Theta(1)$.

3.2.2 Finden des minimalen Schlüssels

Wegen der Min-Heap-Eigenschaft der binomialen Bäume in einem binomialen Heap, muß sich der Knoten mit minimalem Schlüssel in einem der Wurzelknoten befinden. Die Prozedur *BINOMIAL-HEAP-MINIMUM* durchläuft die Wurzelliste und gibt den Wurzelknoten mit kleinsten Schlüssel zurück. Da die Wurzelliste höchstens $\lfloor \lg n \rfloor + 1$ lang ist, ist die Laufzeit in $O(\log(n))$.

3.2.3 Fusionieren zweier binomialer Heaps

In dieser Prozedur namens *BINOMIAL-HEAP-UNION (H1, H2)* werden zwei binomiale Heaps $H1$ und $H2$ zu einem neuen binomialen Heap H vereinigt. Als erstes werden über die Unterprozedur *BINOMIAL-HEAP-MERGE* die Wurzellisten von $H1$ und $H2$ zusammengefügt, und zwar so, daß die neue Wurzelliste wieder streng monoton steigend nach Wurzelgrad sortiert ist. Die Prozedur ist vergleichbar mit der *merge*-Prozedur aus *Mergesort*, nur daß nach Wurzelgrad sortiert wird. Danach durchläuft die Prozedur angefangen beim linkesten Wurzelknoten die Wurzelliste von links nach rechts und fügt Binomialbäume gleicher Größe zusammen, indem die Wurzel des einen Binomialbaumes mit größerem Schlüssel als linkes Kind an die Wurzel des anderen geheftet wird. Zu anfangs können dabei höchstens zwei Binomialbäume gleicher Größe in der neuen Wurzelliste auftauchen. Später, wenn die Prozedur begonnen hat, können als Zwischenstadium drei Binomialbäume gleicher Größe hintereinander auftreten. Dieser Fall tritt dann auf, wenn zwei Binomialbäume B_{k-1} in der gemergten Wurzelliste zu einem B_k zusammengefügt werden, aber zwei weitere B_ks bereits vorhanden waren. Die Prozedur muß diese Fälle unterscheiden und rückt nacheinander von einer Wurzel zur nächsten in der Wurzelliste vor, bis das Ende dieser erreicht ist und der neue binomiale Heap nun für ein ganzzahliges $i \geq 0$ höchstens einen binomialen Baum B_i hat.

Wenn $H1$ n_1 viele und $H2$ n_2 viele Knoten hatte, dann verläuft die Prozedur *BINOMIAL-HEAP-MERGE(H1, H2)* in $O(\log(n))$ mit $n = n_1 + n_2$. Die Wurzelliste von $H1$ ist nämlich höchstens $\lfloor \lg n_1 \rfloor + 1$ lang, für $H2$ gilt entsprechend, daß dessen Wurzelliste höchstens $\lfloor \lg n_2 \rfloor + 1$ lang ist: $\lfloor \lg n_1 \rfloor + 1 + \lfloor \lg n_2 \rfloor + 1 \leq 2 \lfloor \lg n \rfloor + 2 = O(\log(n))$.

Für den Rest der Prozedur *BINOMIAL-HEAP-UNION(H1,H2)* müssen zusätzlich eine konstante Anzahl von Schritten (Vergleichsoperationen, Umlegen von Zeigern) mal Länge der gemergten Wurzelliste, die höchstens $\lfloor \lg n \rfloor + 1$ groß ist, durchgeführt werden. Die Prozedur verläuft also in $O(\log(n))$.

3.2.4 Einfügen eines Knotens

Bei der Prozedur *BINOMIAL-HEAP-INSERT(H,x)* wird ein neuer Heap H' über *MAKE-BINOMIAL-HEAP()* erzeugt, wobei der Zeiger $kopf[H']$ auf x umgelenkt wird. H' enthält nun den Knoten x. Danach wird über $BINOMIAL-HEAP-UNION(H, H')$ x erfolgreich in H eingefügt. *BINOMIAL-HEAP-INSERT(H,x)* läuft in $O(\log(n))$ Zeit.

3.2.5 Entnehmen eines Knotens mit minimalem Schlüssel

In *BINOMIAL-HEAP-EXTRACT-MIN(H)* wird als erstes der Knoten x mit minimalem Schlüssel gesucht, der ja in der Wurzelliste von H zu finden ist. x wird entfernt und die Kinder von x werden nun in umgekehrter Reihenfolge einem allokierten leeren Objekt H' zugefügt. Aus dem Lemma für binomiale Bäume, Eigenschaft 4, wissen wir daß die Kinder der Wurzel eines Baum B_k von links nach rechts die Wurzeln der Bäume $B_{k-1}, B_{k-2}, ..., B_0$ sind. H' ist also ein vollwertiger binomialer Heap, der nun über $BINOMIAL-HEAP-UNION(H, H')$ wieder in den ursprünglichen binomialen Heap eingegliedert wird. Heraus kommt wieder ein binomialer Heap, der Knoten mit minimalstem Element wurde erfolgreich entfernt. Das ganze geht in $O(\log(n))$ Zeit.

3.2.6 Verringern eines Schlüssels

Die Prozedur *BINOMIAL-HEAP-DECREASE-KEY(H,x,k)* verringert den Wert eines Schlüssels eines Knoten x auf einen kleineren Wert k. Dabei bekommt sie einen Zeiger auf x übergeben, dessen Schlüssel auf den Wert von k gesetzt wird. Danach wird sichergegangen, daß der Binomialbaum, in dem x sich befindet, immer noch die *MIN-HEAP*-Eigenschaft hat. Das wird durch Vergleichen des Schlüssels von x mit dem Schlüssel des Vaterknotens erreicht, notfalls wird x mit dem Vaterknoten vertauscht, bis x neuer Wurzelknoten ist. Das alles geschieht zeitlich in Abhängigkeit von der Höhe des binomialen Baumes, in dem x sich befindet. Da wir wissen, daß ein binomialer Baum B_k Höhe k hat und in einem binomialen Heap H mit n Elementen der höchste binomiale Baum $B_{\lfloor \lg n \rfloor}$ ist, verläuft die Prozedur erneut in $O(\log(n))$.

3.2.7 Entfernen eines Schlüssels

In *BINOMIAL-HEAP-DELETE(H,x)* wird der Schlüssel von x auf den Wert $-\infty$ gesetzt, indem $BINOMIAL-HEAP-DECREASE-KEY(H, x, -\infty)$ aufgerufen wird. Danach wird dieser Knoten über *BINOMIAL-HEAP-EXTRACT-MIN(H)* aus der Wurzelliste entfernt. Beide Operationen verlaufen in logarithmischer Zeit, also verläuft auch *BINOMIAL-HEAP-DELETE(H,x)* in $O(\log(n))$.